PROYECTO SISTEMAS ELECTRÓNICOS DIGITALES

CURSO 2021/2022

MEMORIA TRABAJO VHDL

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE UNA MÁQUINA EXPENDEDORA EN FPGA

INTEGRANTES DEL GRUPO:

DAVID HERGUETA SOTO	Nº MAT: 54666
IGNACIO SÁNCHEZ APARICIO	Nº MAT: 54854
DEWEI ZHOU	Nº MAT: 54912

REPOSITORIO:
https://github.com/davidherguetasoto/Proyecto_maquina_expendedora_VHDL.git

ÍNDICE

INTRODUCCIÓN **2**

ENTIDADES **2**

TOP 2

SYNCHRNZR 4

DEBOUNCER 4

EDGEDTCTR 5

CONTADOR 5

FSM 6

- MASTER_FSM 8
- SLAVE_FSM 9

DISPLAY_CONTROL 10

TESTBENCHES **11**

tb_CONTADOR 11

tb_SLAVE_FSM 12

tb_FSM 12

tb_Displays 13

INTRODUCCIÓN

El objetivo del proyecto ha consistido en el diseño, programación en VHDL e implementación en FPGA del sistema de una máquina expendedora.

El usuario podrá seleccionar entre tres productos distintos, y este se venderá cuando se haya introducido la cantidad exacta de un euro.

La máquina será capaz de llevar la cuenta del dinero introducido, y además lo mostrará a través de los displays de 7 segmentos.

Cuando el usuario haya introducido la cantidad exacta de un euro, la máquina indicará con un mensaje escrito en los displays que el producto se está vendiendo con éxito. Además, lanzará un temporizador que, tras finalizar, retornará a la máquina al estado inicial.

Si el usuario introduce una cantidad de dinero mayor a un euro, la máquina entrará en un estado de error, indicándolo de igual manera con un mensaje a través de los displays, y lanzará otro temporizador, que tras finalizar volverá de nuevo al estado inicial con la cantidad de dinero de cero euros, mostrando así que la máquina ha devuelto el dinero introducido.

Para representar que el usuario introduce dinero en la máquina, se han programado cuatro botones en la FPGA que añadirán la cantidad de 10, 20, 50 céntimos, o 1€ al dinero que se haya introducido previamente cada vez que sean pulsados.

El usuario podrá elegir entre los tres diferentes productos a través de los switches de la FPGA. Cada vez que se seleccione uno, se iluminará su LED correspondiente indicando que el producto ha sido seleccionado.

Solamente se podrá seleccionar un producto a la vez. Si hay más de uno seleccionado (o ninguno) y se introduce la cantidad de un euro en la máquina, la máquina no venderá nada.

Se ha programado también un botón con la función de RESET que eliminará los valores de todos los registros internos de la máquina, poniendo de nuevo la cantidad de dinero a cero y la máquina en estado de reposo de manera asíncrona cada vez que sea pulsado. El botón de RESET es activado a nivel bajo.

A continuación, se explicarán con más detalle las entidades que componen el código del sistema.

Todo el código se encuentra subido al repositorio de Github adjunto en la portada de este documento, desde el cual se podrán descargar todos los archivos de código fuente, junto con el de las simulaciones, y el fichero de restricciones. Para poder subir el código a una FPGA se deberá crear un proyecto en VIVADO en el que se incluirán los ficheros de código fuente junto con el de restricciones subidos al repositorio (si se utiliza en una placa distinta de la Nexys 4 DDR, el usuario deberá crear e introducir su propio fichero de restricciones).

ENTIDADES

TOP

La entidad TOP es la principal de nuestro código y que recoge todos los componentes que conforman el sistema.

Cuenta con una entrada por cada botón de dinero (*button_10cent, 20cent, 50cent* y *1euro*), una entrada para la señal de reloj que llevará el sistema, una entrada de un bus de tres bits *producto* (una para cada switch que sirve para seleccionar el objeto a comprar deseado), y una entrada para la señal de *reset*; y como salidas, cuenta una señal de 3 bits denominada *led*, que se encarga de controlar el encendido de los LEDs de la placa que indican el producto seleccionado; y finalmente, para el control de los displays de la FPGA, la entidad tiene dos salidas de dos buses de 8 elementos: *digsel* para el control del encendido de cada uno de los displays, y *segmentos* para el encendido de cada uno de los segmentos que componen los displays.

El acondicionamiento y sincronización con el reloj de las señales procedentes de los botones se realiza con las entidades SYNCHRNZR, DEBOUNCER y EDGEDTCTR. Como solamente manejan 1 bit, se han instanciado 4 componentes de cada una de las entidades antes mencionadas, uno para cada señal y poder así acondicionar las cuatro.

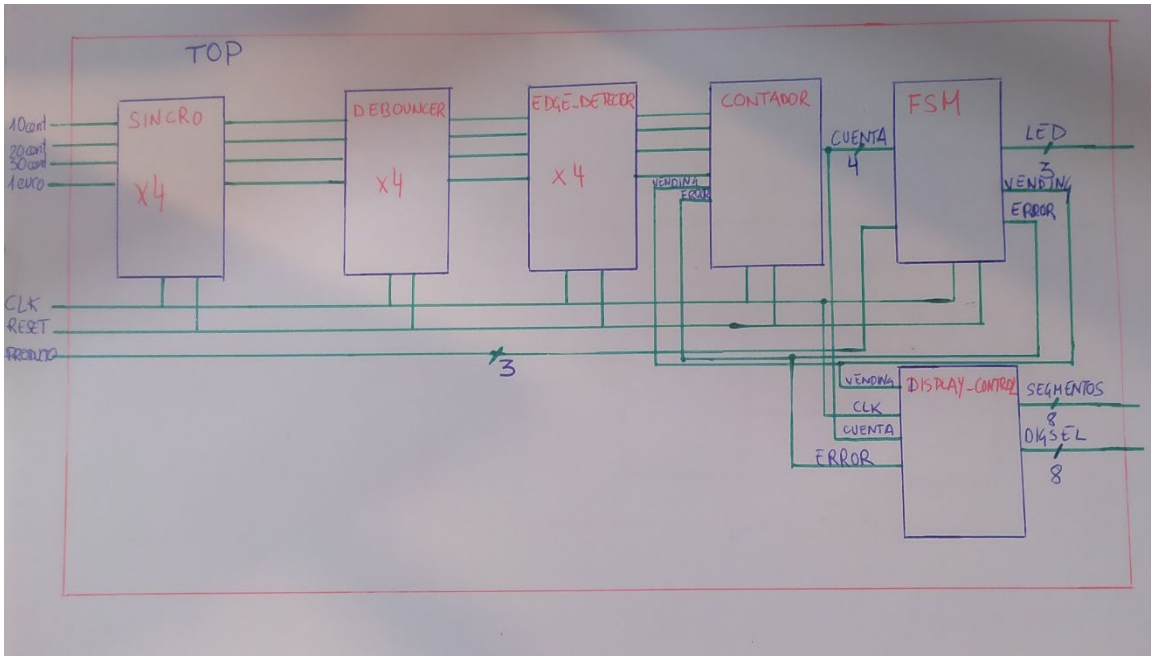

Ilustración 1. Boceto del diagrama de bloques de la entidad TOP.

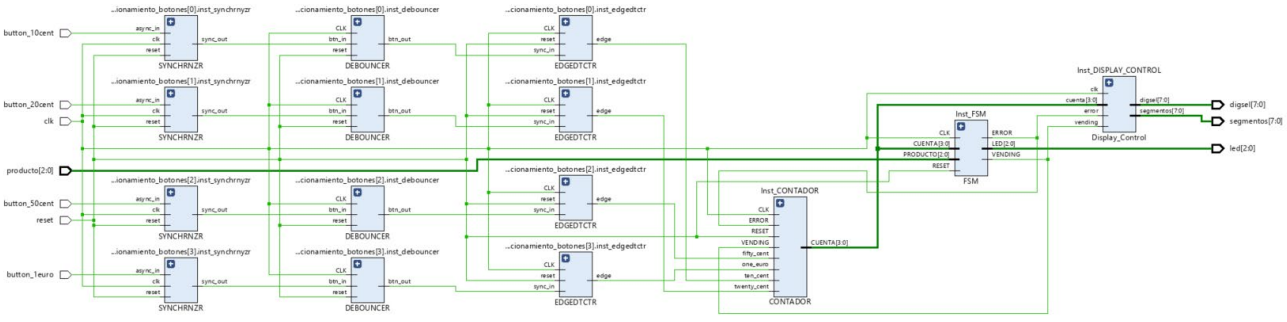

Ilustración 2. Diagrama de bloques de la entidad TOP generado con la vista elaborada del proyecto en Vivado.

La instanciación de cada una de ellas se hará con ayuda de un for generate.

```
acondicionamiento_botones: for i in 0 to 3 generate
inst_synchrnyzr: SYNCHRNZR port map(
                            clk=>clk,
                            async_in=>monedas(i),
                            sync_out=>sync_media(i),
                            reset=>reset);

inst_debouncer: DEBOUNCER port map(
                            CLK=>clk,
                            btn_in=>sync_media(i),
                            btn_out=>deb_media(i),
                            reset=>reset);

inst_edgedtctr: EDGEDTCTR port map(
                            CLK=>clk,
                            sync_in=>deb_media(i),
                            edge=>sal_edge(i),
                            reset=>reset);
end generate acondicionamiento_botones;
--SINCRONIZADORES
```

Ilustración 3. For generate empleado para generar cada uno de los componentes de la entidad.

Se ha creado también la señal *monedas* que es un vector de 4 bits que agrupa las señales de cada uno de los botones para la entrada del dinero. Se introducen en un vector para poder luego usarlo de forma más sencilla en el for generate.

```
--ASIGNACIÓN DE LOS BOTONES DE MONEDAS A CADA POSICIÓN DEL VECTOR
monedas(0)<=button_10cent;
monedas(1)<=button_20cent;
monedas(2)<=button_50cent;
monedas(3)<=button_1euro;
```

Ilustración 4. Asignación de los botones de dinero a las posiciones del vector monedas.

SYNCHRNZR

Se emplearán 4 sincronizadores, uno por cada señal de entrada producida por los botones que simbolizan las diferentes monedas que se introducen en la máquina.

El sincronizador se encarga, como su propio nombre indica, en sincronizar las señales de entrada con la señal de rejoj de la placa, evitando así la metaestabilidad ocurrida cuando se produce un cambio en la señal de entrada durante un flanco del reloj, pues la pulsación del botón se puede producir de forma asíncrona al reloj.

También cuenta con una entrada para el reloj y de reset.

DEBOUNCER

Se requiere para evitar el efecto de los rebotes que se dan al producir las señales con los botones ya que los pulsadores tienen una naturaleza mecánica. Para ello se utilizan flip flops que son capaces de almacenar el estado anterior, si durante un periodo de tiempo amplio el valor actual concuerda con ese valor almacenado se puede asumir que el estado es estable. Al igual que el sincronizador se repite la estructura 4 veces, una por cada botón.

También cuenta con una entrada para la señal de reloj y otra para la de reset. La realización en cuanto a disposición en la entidad top es la misma que en el caso del sincronizador, instanciamos 4 veces la entidad debouncer.

EDGEDTCTR

La pulsación mecánica de los botones no es de una duración determinada, por lo que se pueden encontrar inconvenientes a la hora de registrar la señal introducida. Este módulo se emplea para que cada vez que se produzca un flanco de bajada en el pulsador, la señal de salida del módulo se active, contando así solamente los flancos del pulsador, evitando así que se produzcan cambios continuamente mientras el botón se encuentra pulsado.

Al igual que los dos módulos anteriores este también se empleará cuatro veces, una por cada señal de botón.

También cuenta con una entrada para la señal de reloj y otra para la de reset.

CONTADOR

La entidad CONTADOR es la encargada de sumar y llevar la cuenta del dinero introducido en la máquina a través de sus entradas.

En su interior contiene un registro de 5 bits (*cuenta_aux*) que será el que acumule a cantidad de dinero. Si la entrada *ten_cent* se activa, sumará 1 unidad a la cuenta; si lo hace *twenty_cent* sumará 2 unidades; si lo hace *fifty_cent* sumará 5 unidades, y si lo hace *one_euro* sumará 10 unidades. Todo esto lo realizará de forma síncrona.

Las entradas *ERROR* y *VENDING* le permiten conocer si la máquina ha entrado en estado de error, o se encuentra vendiendo un producto respectivamente. En cualquiera de los dos casos, si una de estas dos entradas se activa, la cuenta se reiniciará a cero igual de manera síncrona.

El resultado de la cuenta será transmitido a la salida CUENTA de 4 bits, siguiendo la siguiente tabla de equivalencia:

DINERO	cuenta_aux	CUENTA
0€	00000	0000
0.1€	00001	0001
0.2€	00010	0010
0.3€	00011	0011
0.4€	00100	0100
0.5€	00101	0101
0.6€	00110	0110
0.7€	00111	0111
0.8€	01000	1000
0.9€	01001	1001
1€	01010	1010
>1€	otros valores	1011

El registro interno cuenta con un bit más que la salida con el objetivo de no producirse desbordamientos a la hora de realizar sumas, por que en caso de que en una suma el bit más significativo de *cuenta_aux* salga 1, la cuenta se realizaría con éxito sin desbordarse, en la salida *CUENTA* se escribiría 1011, y al detectarlo, en el siguiente ciclo de reloj la entrada *ERROR* se activaría, produciendo que *cuenta_aux* vuelva a 0, pudiendo seguir haciendo sumas sin problemas.

Si solo tuviese 4 bits, la suma de 1€ + 1€ no podría hacerse, pues sería 01010 + 01010 = 10100, un número de 5 bits en el que MSB es 1 por lo que con 4 provocaría desbordamiento.

Si la entrada *RESET* se activa (activa a nivel bajo), la salida *CUENTA* y el registro *cuenta_aux* pasarán a ser 0 de forma asíncrona y con prioridad.

FSM

Será la entidad encargada de implementar el funcionamiento de la máquina de estados del sistema.

Se compone de dos submáquinas de estado (esclava y maestra), para poder llevar a cabo tareas de temporización en algunos puntos del proceso.

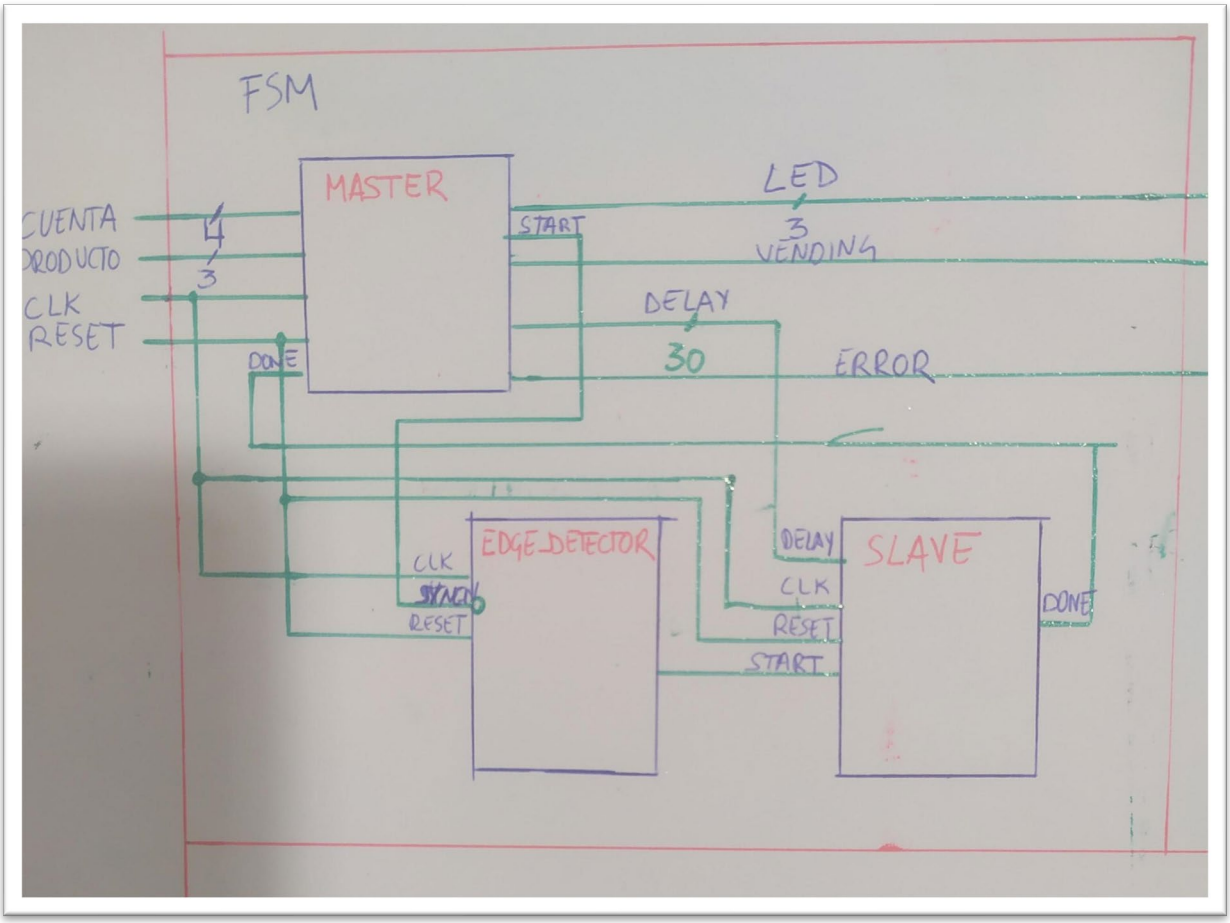

Ilustración 5. Boceto del diagrama de bloques de la entidad FSM.

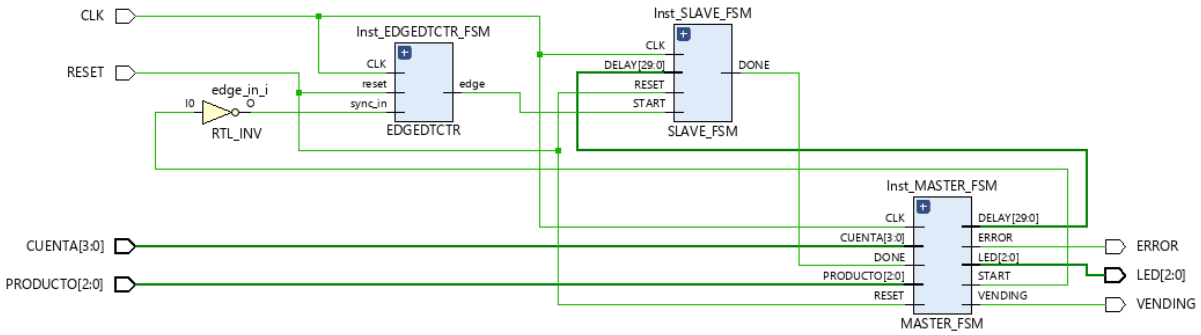

Ilustración 6. Diagrama de bloques de la entidad FSM generado con la vista elaborada del proyecto en Vivado.

La submáquina maestra será la encargada de los cambios de estados y los cambios de las señales de salida de la FSM, mientras que la máquina esclava se encargará de las tareas de temporización.

La submáquina esclava iniciará un temporizador con el tiempo que le pase la maestra a la esclava en la salida DELAY, cuando detecte un flanco ascendente en la señal START. Si la cuenta termina, la esclava introducirá a la maestra un '1' lógico en la señal DONE indicando que el tiempo ha finalizado.

Los estados de la FSM están representados en el siguiente diagrama:

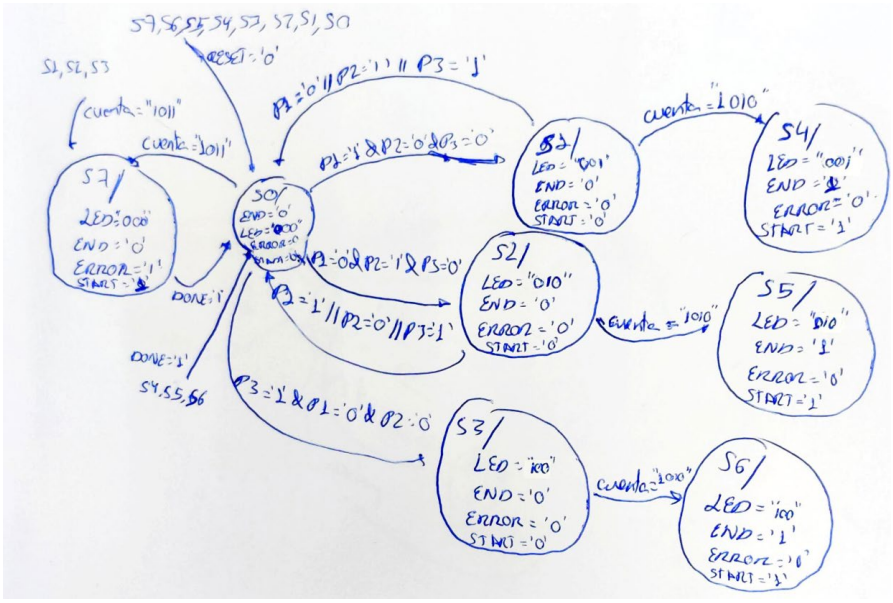

Ilustración 7. Boceto de la máquina de estados del sistema.

La máquina comenzará en el estado de reposo S0 en el que todas las salidas de la máquina se encontrarán inactivas en el valor lógico '0'.

Estando la máquina en el estado de reposo, si se activa a través de los switches una entrada de PRODUCTO para seleccionar un producto, la máquina pasará a otro estado en el que se activará el LED ligado a la selección de dicho producto, esperando a que se introduzca la cantidad de dinero correcta.

Si se selecciona más de un producto a la vez, la máquina no pasará del estado de espera S0, y en caso de que se hubiera seleccionado previamente un producto, y estuviese en el estado S1, S2 o S3, si se

selecciona otro producto, volvería al estado de espera S0. De esta forma se asegura que la máquina venda un producto si solamente se encuentra uno seleccionado.

Si estando en S1, S2 o S3 se introduce la cantidad de 1€ en la máquina expendedora (CUENTA=”1010”), la FSM pasará a otro estado en el que se indicará que el producto se ha vendido correctamente activando la salida END con un ‘1’ lógico. Dependiendo del producto que se haya vendido, se activará la salida de su LED correspondiente.

Estando en el estado S4, S5 o S6, se activará la cuenta de un temporizador de 5 segundos, que al finalizar activará la entrada DONE. Cuando esta señal se active, la FSM volverá al estado S0 de reposo.

Si desde los estados S0, S1, S2 o S3 la máquina expendedora detecta que se ha introducido una cantidad de dinero mayor a 1€ (CUENTA=”1011”), la máquina pasará al estado S7 de error, en el que se activará con un ‘1’ lógico la salida ERROR de la FSM, y comenzará a correr otro temporizador de 5 segundos.

Una vez que el temporizador acabe, la entrada DONE se activará, y la máquina volverá al estado S0 de reposo.

Si desde cualquier estado se activa la entrada de RESET (a nivel lógico bajo), la máquina retornará al estado de reposo S0.

*NOTA: Todas las señales con la coletilla _viewer se emplearon para extraer la evolución de todas las señales internas de la entidad durante la simulación. No son necesarias para el normal funcionamiento del sistema.

- MASTER_FSM

Su código se compone de tres procesos distintos: uno para poder cambiar al estado siguiente en cada ciclo de reloj, o en caso de que se haya recibido una señal de RESET, volver al estado de reposo de forma asíncrona y prioritaria (*ACTUALIZACION_DE_ESTADO*); otro MASTER_FSM (*CAMBIO_DE_ESTADI*); y finalmente, otro para controlar las salidas de la entidad en función del estado en el que se encuentre.

Los estados que contiene la máquina se indican con el tipo de datos enumerado *STATE*, podrá tomar los valores *S0, S1, S2, S3, S4, S5, S6* y *S7*.

Para controlar los estados de la máquina, se han creado las señales *present_state* que representa el estado en el que se encuentra actualmente la máquina, y que por defecto se inicializará en el estado de reposo; y *next_state* que servirá para almacenar el estado al que se debe avanzar en el siguiente ciclo de reloj. Ambas son del nuevo tipo de datos creado *STATE*.

```
type STATE is (
    S0, --ESTADO DE REPOSO
    S1, --PRODUCTO 1 SELECCIONADO
    S2, --PRODUCTO 2 SELECCIONADO
    S3, --PRODUCTO 3 SELECCIONADO
    S4, --VENDIENDO PRODUCTO 1
    S5, --VENDIENDO PRODUCTO 2
    S6, --VENDIENDO PRODUCTO 3
    S7  --ESTADO DE ERROR
    );
signal next_state : STATE;
signal present_state : STATE:=S0;
```

Ilustración 8. Tipo de datos STATE junto con las señales present_state y next_state de la MASTER_FSM.

La entidad posee también dos constantes *ERROR_DURATION* y *VENDING_DURATION* de tipo positivo, en las que el programador podrá introducir el tiempo que desea que la máquina se encuentre en el estado de ERROR o de venta respectivamente.

```
constant ERROR_DURATION : positive :=500000000; --DURACIÓN DE ESPERA TRAS ESTADO DE ERROR EN CICLOS DE RELOJ
constant VENDING_DURATION: positive:=500000000; --DURACIÓN DE ESPERA TRAS VENTA
```

Ilustración 9. Constantes ERROR_DURATION y VENDING_DURATION de la MASTER_FSM.

Como la FPGA empleada en el proyecto cuenta con un reloj de 100MHz y queremos que en ambos estados se pare un tiempo de 5 segundos, les hemos introducido un valor de 500000000 a cada una de las constantes.

Estas constantes posteriormente se introducirán en la salida *DELAY* (convirtiéndolas previamente al tipo de datos UNSIGNED) para indicarle a la SLAVE_FSM encargada de la temporización, el tiempo que debe permanecer contando.

- SLAVE_FSM

La máquina slave se encargará de llevar la cuenta de los temporizadores.

La cuenta comenzará cuando se produzca un flanco positivo en la señal de entrada *START*. Para la detección de los flancos, se ha introducido un componente EDGEDTCTR entre la salida *START* de la MASTER_FSM, y la entrada *START* de la SLAVE_FSM.

Como la entidad EDGEDTCTR detecta flancos negativos de su señal de entrada, se le ha añadido una puerta NOT a la salida *START* de la MASTER_FSM para que así la detección de flancos sea con los ascendentes.

Una vez producido el flanco ascendente en *START*, se carga en la variable *count* el tiempo que haya en la entrada *DELAY*, y se pondrá la variable *aux_start* a '1' y *aux_done* a '0'.

La señal *START* sirve para poder inicializar el temporizador, y el temporizador solamente contará cuando la señal *aux_start* se haya puesto a '1', lo que indicará que en algún momento anterior la entrada *START* ha sido activada. La cuenta se irá decrementando en una unidad con cada ciclo de reloj en ese caso.

La señal *aux_done* se encargará de controlar la salida *DONE.* DONE se activará cuando la cuenta del temporizador haya terminado.

Cuando la señal *aux_start* esté activada, y la cuenta llegue a cero, se activará la señal *aux_done,* lo que indica que la cuenta previamente se le había ordenado comenzar, y acaba de terminar.

Si antes de eso, la señal *aux_start* se encontraba a cero, la señal *aux_done* permanecerá inactiva, pues eso significaría que la cuenta no se le había ordenado comenzar.

El objetivo de este algoritmo era el de subsanar errores que ocurrían previamente con el temporizador en los cuales o nunca se activaba la señal *DONE*, o directamente se activaba la salida *DONE* al detectar un flanco en *START* sin haber comenzado la cuenta (pues al inicio, la señal *cuenta* se encuentra a cero, y lo detectaba entendiendo que ya había terminado la cuenta, y activaba *DONE* directamente sin realizar la cuenta).

```
process (CLK, RESET)
begin
  if RESET = '0' then
    count <= (others => '0');
    aux_done<='0';
  elsif rising_edge(CLK) then
    if START='1' then
      count <= DELAY;
      aux_done<='0';
      aux_start<='1';
    end if;

    if aux_start='1' then
       if count /= 0 then
         count <= count - 1;
         aux_done<='0';
       elsif count=0 then
        aux_done<='1';
         aux_start<='0';
       end if;
    elsif aux_start='0' then
        aux_done<='0';
    end if;
  end if;
end process;

DONE<=aux_done;
```

Ilustración 10. Algoritmo del temporizador de SLAVE_FSM.

DISPLAY_CONTROL

La entidad Display_Control es la encargada de mostrar en los displays la cuenta del dinero introducido en la máquina y los mensajes de *error* o el *sold*.

Cuenta con una entrada de 4 bits que indica lo que se quiere mostrar en los diplays (*cuenta*), una entrada de 1 bit (*vending*), que muestra la palabra "*sold"* en los displays cuando es '1', otra entrada de de 1bit (*error*) que muestra la palabra "*error*" en los displays cuando es '1', y otra entrada para el reloj (*clk*). También cuenta con 2 salidas: la salida *digsel* de 8 bits que indica cuáles de los 8 displays de la placa se van a encender, y la salida *segmentos* de 8bits (7segmentos + el punto) que indica qué segmentos del display se van a encender.

Debido a que los cátodos de los segmentos son comunes a los ocho displays, se mostraría el mismo digito en cada display. Para que en los displays no muestren lo mismo, los displays se manejarán en periodos de 1.6ms, es decir, como tenemos 8 displays, cada display se ilumina durante 1/8 periodo de forma consecutiva. Por ejemplo, si se quiere mostrar en los displays el número 71, lo que se haría es encender primero el digito "1" durante 1/8 periodo (0.2ms), transcurrido ese tiempo, se enciende el digito "7" durante otros 1/8 periodo, y esto se repetiría cada periodo. Como se repite lo mismo cada 1.6ms, se verá que el número 71 estará encendido continuamente en los displays.

Para implementarlo en el código se ha utilizado un signal *clk_counter* que cuenta el número de flanco de subida del reloj (*clk*), como el reloj es de 100MHz, el número de flancos que tiene que contar para un periodo de 1.6ms es 160000, y para 1/8 periodo el *clk_counter* tiene que contar 20000 flancos de subida, lo que equivale a 0.2ms. Lo que se quiere hacer es que cada vez que se cuente hasta 20000, se incrementa el *signal anodos* que será el encargado de indicar el display que se quiere encender, por ejemplo, si es 0 se encienda el primer display y si es 1 se enciende el segundo display. En la siguiente figura se muestra el primer process:

```
process(clk)
 begin
     --Periodo 1.6 ms-> clk_counter=160000
    if rising_edge(clk) then
      clk_counter<=clk_counter + 1;

      if anodos > 7 then
         anodos<=0;
      end if;

       --periodo/8 = 0.2 ms -> clk_counter=20000
      if clk_counter>=20000 then
         clk_counter<=0;
         anodos<=anodos +1;
      end if;
    end if;

 end process;
```

Ilustración 11. Contador de flanco de subida de clk

En el segundo process se encargará de encender los displays según el valor de *numero*, *vending* o *error*. El valor de la entrada *cuenta* se pasa a entero y se guarda en signal *numero*. Por ejemplo, si *numero* es '3' se encenderán los tres primeros displays (durante 1/8 periodo cada uno), el resto estarán apagados. En el tercer process indicará qué se mostrará en los displays según el valor de *numero*, *vending* o *error*. Por ejemplo, si *numero* es '3', se mostrará en los displays "0.30", para ello utilizamos el signal *show* que indicará que segmentos se tiene que encender en el siguiente process. En el último process indicará los segmentos que se van a activar en el display según el valor de *show.*

TESTBENCHES

Para comprobar el correcto funcionamiento de las entidades, se han realizado testbenches sobre aquellas que pudieran resultar más conflictivas a la hora de programar, con los que se podían subsanar fácilmente errores que quedaban visibles en los diagramas de tiempos de las señales y que producían fallos al implementar el código en FPGA.

Se explicará con detalle cada uno a continuación junto con el diagrama de tiempo de la simulación.

tb_CONTADOR

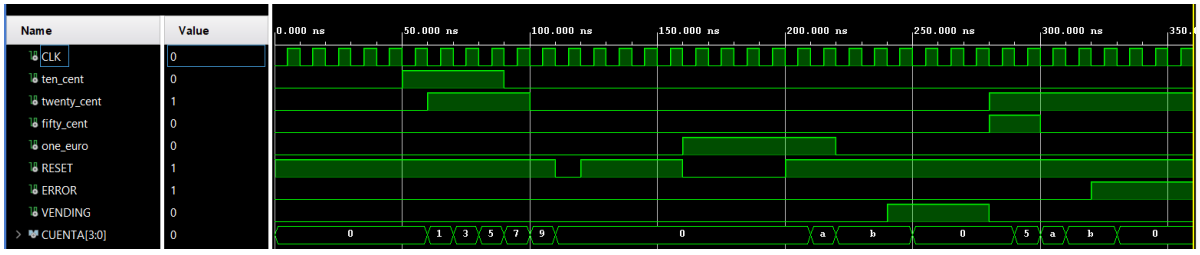

Ilustración 12. Diagrama de tiempos de la simulación tb_CONTADOR.

Se han simulado 350ns con un reloj de 100MHz. Se observa que el contador funciona correctamente. Añade a la cuenta el valor de la señal que más dinero tenga asignada y que se encuentre activa. Se

reinicia a cero si *ERROR* o *VENDING* se encuentra activa, y se resetea de forma asíncrona cuando *RESET* pasa a ser '0'.

tb_SLAVE_FSM

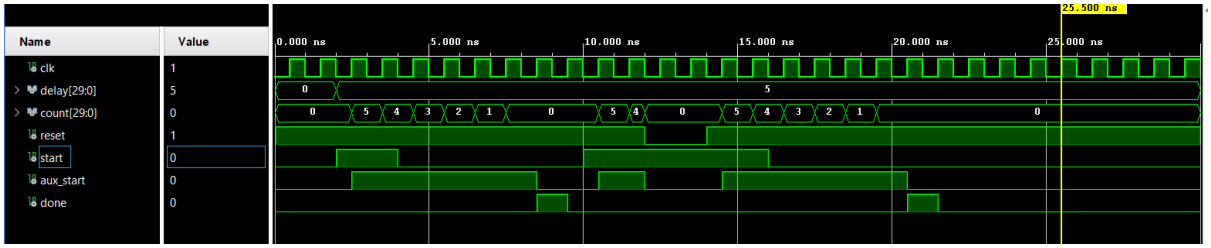

Ilustración 13. Diagrama de tiempos de la simulación de tb_SLAVE_FSM.

El temporizador funciona correctamente. Se ha simulado con un reloj de 1GHz y una simulación de 30ns.

El temporizador carga correctamente el valor de la señal *DELAY* en la cuenta cuando *START* pasa a '1', y decrece su valor en 1 con cada ciclo de reloj.

Cuando llega la cuenta a cero, se activa bien la salida *DONE*.

Si *RESET* pasa a '0', la cuenta pasa a cero, y la señal de *aux_start* también pasa a '0'.

tb_FSM

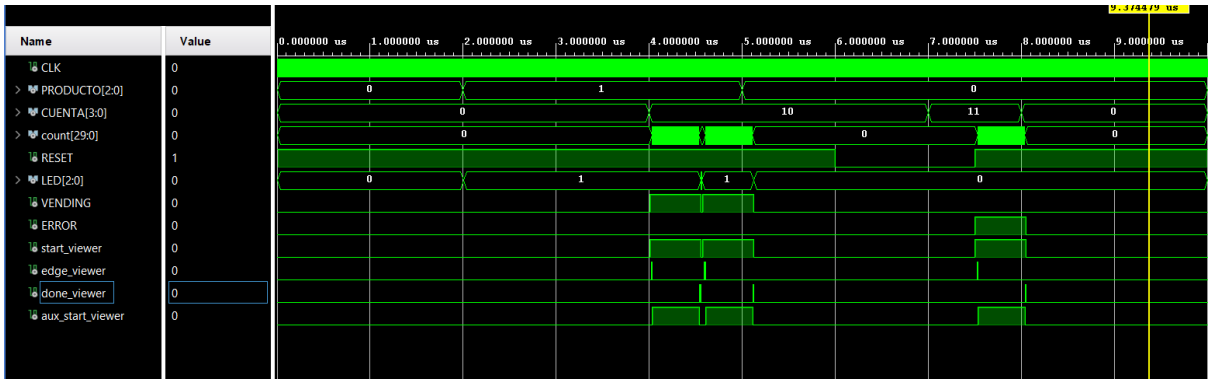

Ilustración 14. Diagrama de tiempos de la simulación de tb_FSM.

Simulación de la máquina de estados con un reloj de 100MHz, y un tiempo de simulación de 10 microsegundos.

Las constantes *ERROR_DURATION* y *VENDING_DURATION* se han adaptado al tiempo de simulación para que el tiempo que la máquina permanece en estado de error y el tiempo en el que permanece vendiendo el producto sea el adecuado al tiempo del diagrama.

```
constant ERROR_DURATION : positive :=50; --DURACIÓN DE ESPERA TRAS ESTADO DE ERROR EN CICLOS DE RELOJ
constant VENDING_DURATION: positive:=50; --DURACIÓN DE ESPERA TRAS VENTA
```

Ilustración 15. Duración del tiempo de error y venta de la máquina ajustado para la simulación.

En el diagrama se observa que la máquina funciona correctamente.

La señal *CUENTA* indica el dinero que se ha introducido en la máquina, y la señal *count* indica la cuenta del temporizador de la máquina esclava.

Comienza a contar solamente cuando hay un producto seleccionado, y hay una cantidad de 1€ en la máquina (*CUENTA* = 1010).

Una vez la cuenta termina, se produce un flanco en la señal *done* de la máquina esclava. Cuando eso se produce, la máquina vuelve al estado de reposo, pero como hay un producto seleccionado vuelve al estado S1, y como hay 1€ introducido, pasa a S2 directamente y comienza otra cuenta.

Si *CUENTA* vale 1011 (cantidad de dinero introducida mayor a 1€), la máquina pasa a estado de error. Se activa la señal *ERROR*, y comienza de nuevo la cuenta. Cuando termina, vuelve al estado de reposo S0.

La señal *aux_start* se puede observar que solamente permanece activa cuando el temporizador está contando.

Si *RESET* pasa a '0', todas las señales intermedias se reinician, y la cuenta del temporizador vuelve también a cero.

tb_Displays

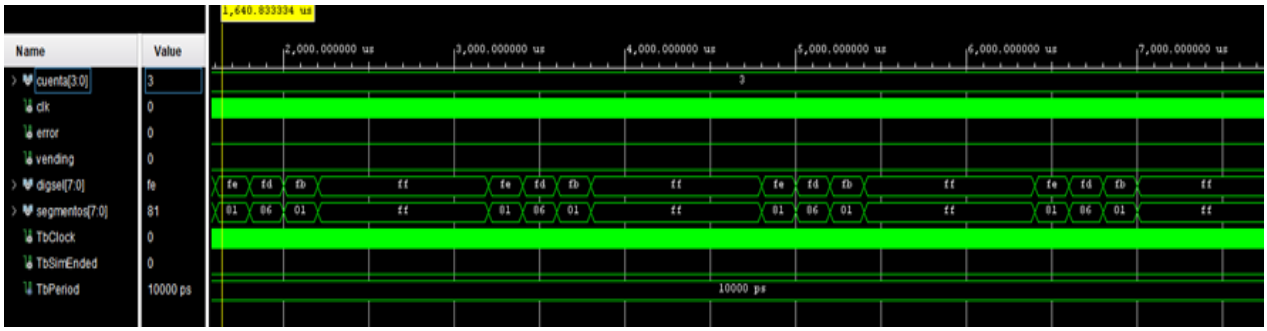

Ilustración 16. Diagrama de tiempos de la simulación de tb_Display cuando cuenta es '3'.

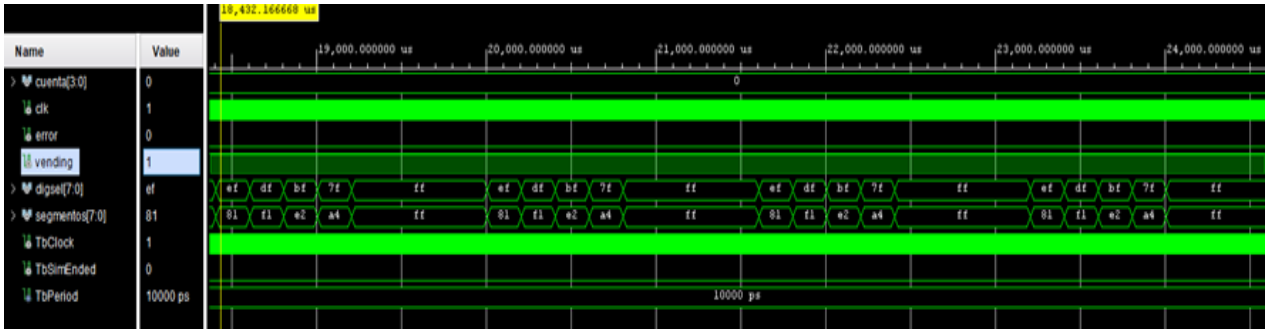

Ilustración 17.Diagrama de tiempos de la simulación de tb_Display cuando vending es '1'.

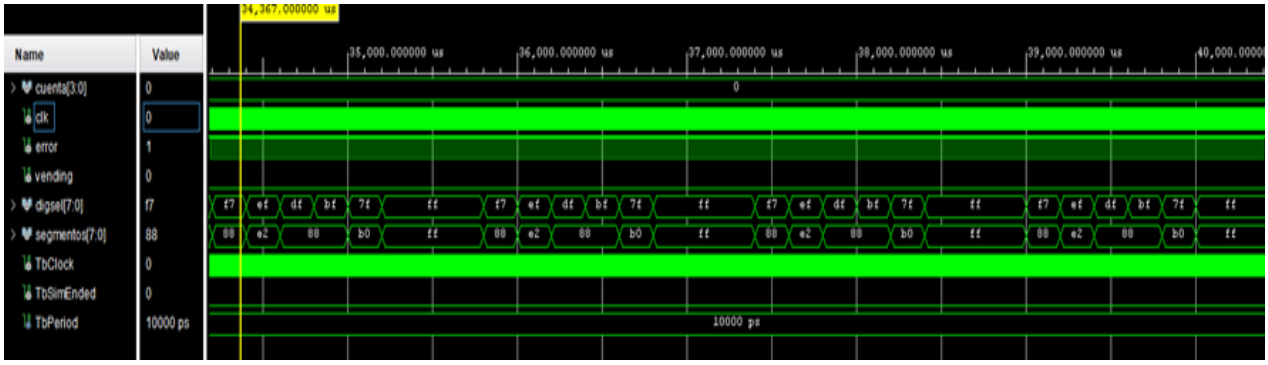

Ilustración 18. Diagrama de tiempos de la simulación de tb_Display cuando error es '1'

Simulación de Display con un reloj de 100MHz, y un tiempo de simulación de 50 milisegundos, teniendo en cuenta que trabajaremos con periodos de actualización de displays de 1.6ms.

Se puede ver qué display se va a encender según el valor de *digsel* y los *segmentos* del display que se van a encender durante 0.2 ms. En el diagrama se puede ver que cuando *cuenta* es '3', en los display de la placa se querrá mostrar "0.30", por lo tanto se encenderá los 3 primeros displays, el resto estarán apagados, cada display se encenderá durante 0.2ms. Cuando el valor de *cuenta* o *segmentos* es "ff" quiere decir que está apagado, como en este caso tenemos 5 diplays apagados, en cada periodo de 1.6ms se puede ver que durante 1ms no se enciende ningún display. Si vending es '1' se querrá mostrar en los displays de la placa la palabra "sold", por lo tanto, 4 de los 8 displays se encenderán durante 0.2ms cada display. Lo mismo pasa cuando *error* es '1' se querrá mostrar en los displays de la placa la palabra "error", por lo tanto, 5 de los 8 displays se encenderán durante 0.2ms cada display.